High-performance computational fluid dynamics: a custom-code approach

High-performance computational fluid dynamics: a custom-code approach

2016 Eur. J. Phys. 37 045001

(http://iopscience.iop.org/0143-0807/37/4/045001)

IOP Publishing European Journal of Physics
Eur. J. Phys. **37** (2016) 045001 (23pp) doi:10.1088/0143-0807/37/4/045001

High-performance computational fluid dynamics: a custom-code approach

James Fannon[1,2,7], Jean-Christophe Loiseau[3], Prashant Valluri[4], Iain Bethune[5] and Lennon Ó Náraigh[1,6]

[1] School of Mathematics and Statistics, University College Dublin, Belfield, Dublin 4, Ireland
[2] School of Physics, University College Dublin, Belfield, Dublin 4, Ireland
[3] Department of Mechanics, KTH Royal Institute of Technology, SE-100 44 Stockholm, Sweden
[4] EPCC, The University of Edinburgh, Edinburgh EH9 3FB, UK
[5] Institute for Materials and Processes, School of Engineering, The University of Edinburgh, Edinburgh EH9 3BF, UK
[6] Complex & Adaptive Systems Laboratory, University College Dublin, Belfield, Dublin 4, Ireland

E-mail: onaraigh@maths.ucd.ie

Received 24 November 2015, revised 2 March 2016
Accepted for publication 31 March 2016
Published 20 April 2016

Abstract

We introduce a modified and simplified version of the pre-existing fully parallelized three-dimensional Navier–Stokes flow solver known as TPLS. We demonstrate how the simplified version can be used as a pedagogical tool for the study of computational fluid dynamics (CFDs) and parallel computing. TPLS is at its heart a two-phase flow solver, and uses calls to a range of external libraries to accelerate its performance. However, in the present context we narrow the focus of the study to basic hydrodynamics and parallel computing techniques, and the code is therefore simplified and modified to simulate pressure-driven single-phase flow in a channel, using only relatively simple Fortran 90 code with MPI parallelization, but no calls to any other external libraries. The modified code is analysed in order to both validate its accuracy and investigate its scalability up to 1000 CPU cores. Simulations are performed for several benchmark cases in pressure-driven channel flow, including a turbulent simulation, wherein the turbulence is incorporated via the large-eddy simulation technique. The work may be of use to advanced undergraduate and graduate students as an introductory study in CFDs, while also providing insight for those interested in more general aspects of high-performance computing.

[7] Present address: Department of Mathematics and Statistics, University of Limerick, Ireland.

S Online supplementary data available from stacks.iop.org/ejp/37/045001/mmedia

Keywords: computational physics, computational fluid dynamics, turbulence, parallel computing

(Some figures may appear in colour only in the online journal)

1. Introduction

Two phase level set (TPLS) is an accurate, highly efficient two-phase Navier–Stokes (NS) solver, parallelized and scalable up to 1000s of CPU cores [1]. The code is available for research-level production runs as open-source software [2]. The development of TPLS was motivated by open questions in the two-phase flow literature (e.g. [3]), which may not be of concern to a student seeking to develop basic skills in computational fluid dynamics (CFDs). Therefore, the aim of the present work is to introduce a simplified (single-phase) version of TPLS and to expose students to high-performance computing through the medium of a classical problem in hydrodynamics—namely fully developed turbulence in single-phase pressure-driven channel flow. In this way, the present work illustrates how contemporary research can inform the understanding of physics at university level. These are ambitious goals, and the article is therefore aimed at students who have taken at least a first course in both mechanic and computational science, although it can be noted that these background topics are already dealt with in pedagogical articles, e.g. [4–7]. We should also emphasize that this article is based on our own experiences of introducing students to the S-TPLS solver, which has in the past culminated in successful undergraduate projects and integration of the flow solver into teaching at the MSc level. In this introduction, we outline some particular features of TPLS, as well as further placing the above physical problems in the context of the broader literature on CFDs.

The full research-level version of TPLS [2] is unique in several aspects. The TPLS solver has been custom-built for supercomputing architectures with large scale simulations of complicated interfacial fluid flows in mind. The full research-level version exploits parallel libraries that are typically available at supercomputing centres, such as PETSc [8] and NetCDF [9]—see [10]. In order to present a highly portable version of the code to students, the methodology presented in this work strips back some of this complexity. The result is a (single-phase) code capable of being run on desktop computers, clusters, and supercomputers —referred to throughout this work as S-TPLS (for simplified-TPLS). Some degradation in the code's performance is expected as a result of this simplification, but the payoff in terms of simplicity and portability is considerable. Performance analysis of the simplified code is addressed in this paper in section 4.

Concerning the physical problem discussed herein, simulation and modelling of turbulent flow provides an attractive problem for students, with the beauty, complexity and infamy of turbulence itself acting as a strong combination of motivating factors. Heuristically, a turbulent flow is characterized by the nonlinear development of eddies (particular patches of fluid illustrating coherent motion) on a wide range of length scales, and velocity vector and pressure fields which are subject to random fluctuations in both time and space, although more formal definitions can be given in terms of some underlying properties [11, 12]. The impasse in terms of an analytical understanding of turbulence (i.e. the open problem of

existence and smoothness of solutions to the NS equations [13]) has motivated a numerical approach to the problem in order to gain further insight. However, one encounters computational restrictions when attempting to simulate turbulent flow accurately, as such a simulation must resolve the fluid motion across all pertinent length scales, which leads to the requirement that the total number of grid points N_T needed in the simulations scales as [14]

$$N_T \sim Re_\ell^{9/4}, \qquad Re_\ell = \frac{\rho U \ell}{\mu}, \tag{1}$$

where Re_ℓ is the Reynolds number based on the large-scale eddies, ρ is the fluid density, U is a characteristic velocity scale, and μ is the dynamic viscosity of the fluid. Typically, ℓ will be comparable to the size of the fluid domain itself and U will be set by the large-scale forcing driving the turbulence. Many physical problems of interest have large values of Re_ℓ, the simulation of which has a huge computational cost.

The computational cost of such direct numerical simulations motivates the introduction of the large-eddy simulation technique (see section 2). This can be regarded as an approximation to direct numerical simulation wherein only turbulent eddies on the largest scales containing the bulk of the system energy are simulated directly, while the smaller eddies are left unresolved and are instead modelled (parametrized). Hence, the computational restriction (1) is lifted, although parametrization of the unresolved eddies in the simulation is a highly non-trivial problem. In the present work the standard Smagorinsky parametrization is used, which is effective for pipe and channel geometries [15].

However, a large-scale LES still presents a considerable computational challenge and thus such a simulation is generally split between many CPU cores. In the case of this work, both a local cluster and a supercomputer (Fionn, ICHEC) were used for the simulations, with the local cluster proving adequate for almost all the requirements of the work. In this way, the LES problem provides a natural introduction to the area of parallel computing, an area of increasing importance for graduate and research work in mathematics and physics. Indeed, using a simplified version of TPLS that can be ported across a variety of architectures, it is possible to introduce in a non-trivial way the topic of high-performance computing to undergraduates in mathematics and the physical sciences in a single semester. The discussion naturally leads to more advanced topics, such as performance analysis and parallel efficiency of the computational code.

The article is organized as follows. Notwithstanding the assumed background knowledge, the acquisition by students of a deep understanding of the fundamental physics and mathematics of fluid dynamics is a driving force in this work. Hence, appropriate mathematical models are constructed in detail in section 2, to encompass both the NS equations and the relevant LES methodology. The models are then solved using numerical approximation. In this work, this is done using the S-TPLS framework, which is introduced in section 3. The performance of the solver is rigorously analysed in section 4. Results of the LES are described in section 5. Finally, concluding remarks are provided in section 6, together with a discussion that places the present computational methodology in the broader context of CFD education.

2. Problem statement

In this section we write down the equations solved by S-TPLS. These are the NS equations both with and without a large-eddy simulation model. We describe the physical scenarios to which these equation sets apply.

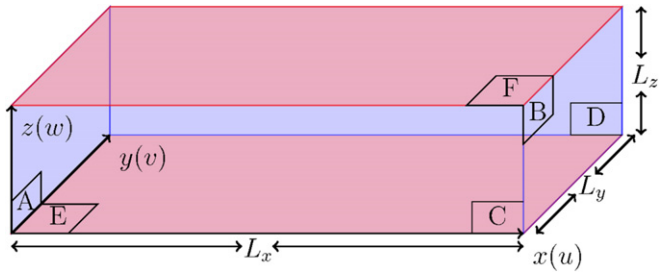

Figure 1. Computational domain $\Omega = (0, L_x) \times (0, L_y) \times (0, L_z = 1)$. Periodic boundary conditions are used in the streamwise (faces *AB*) and spanwise (faces *CD*) directions, while no-slip boundary conditions are used on faces *E* and *F*. Image from [10] by the present authors.

2.1. NS equations

We consider an incompressible, Newtonian fluid confined in a channel geometry $\Omega = (0, L_x) \times (0, L_y) \times (0, L_z)$ subject to a constant negative pressure gradient $\mathrm{d}p/\mathrm{d}x$ in the x direction. In the absence of gravity and other external forces, the non-dimensional NS equations can be written in component form as

$$\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{1}{Re_*} \frac{\partial^2 u_i}{\partial x_j \partial x_j}, \tag{2a}$$

$$\frac{\partial u_i}{\partial x_i} = 0, \tag{2b}$$

where the Einstein summation convention is used, u_i denotes the velocity-field components $\boldsymbol{u} = (u, v, w) \equiv (u_1, u_2, u_3)$, and p denotes the pressure field. The quantity Re_* is the Reynolds number $Re_* = \rho U L_z/\mu$ based on the velocity scale

$$U = \sqrt{\frac{L_z}{2\rho} \left| \frac{\mathrm{d}p}{\mathrm{d}x} \right|}. \tag{3}$$

Equation (3) defines the so-called friction velocity: ρU^2 gives the wall shear stress at the top and bottom walls of the channel—see [14]. Finally, in this non-dimensional scheme, the non-dimensional channel height is set to unity, $L_z \to 1$. A more in-depth introduction to the NS equations can be found in [4].

The computational domain Ω is illustrated in figure 1. The velocity field is subject to periodic boundary conditions in the streamwise (x) and spanwise (y) directions, while no-slip boundary conditions are enforced at the walls i.e. $\boldsymbol{u}(x, y, z = 0, t) = 0 = \boldsymbol{u}(x, y, z = 1, t)$. Periodic boundary conditions in the streamwise and spanwise directions are also applied to the pressure field, modulo the constant pressure drop $\mathrm{d}p/\mathrm{d}x$ applied in the x-direction. This amounts to a constant linear forcing of the flow, which sustains the mean flow in the x-direction. The use of periodic boundary conditions in the streamwise and spanwise directions is appropriate provided that the size of the computational domain in these directions is such that the largest-scale eddies in the flow can be accommodated [16]. This is checked *a posteriori* when simulation results are postprocessed. As such, periodic boundary conditions serve as an accurate model for approximating large systems, of which the present system (flow in long, wide channel) is a good example.

2.2. LES technique

The LES method of simulating turbulent flow has been in existence for over 40 years [17] and is covered in detail in many textbooks [12, 14, 18]. The essential idea is the following: instead of trying to resolve the entire velocity and pressure fields in a simulation, one solves for the so-called filtered velocity and pressure fields which describe the fluid motion exactly down to a given filter width. Small-scale structures which exist below this filter width are thus not resolved in the LES, but their effect on the rest of the flow must be modelled. This process allows for the large-scale structures present in the flow to be resolved while avoiding the high computational cost associated with resolution of small-scale structures.

In order to obtain the dynamic equations which describe the filtered velocity and pressure fields, we apply a filtering process to equation (2) by forming a convolution with a filter function. The filtered equations of motion are

$$\frac{\partial \overline{u_i}}{\partial t} + \overline{u_j}\frac{\partial \overline{u_i}}{\partial x_j} = -\frac{\partial \overline{p}}{\partial x_i} + \frac{1}{Re_*}\frac{\partial^2 \overline{u_i}}{\partial x_j \partial x_j} - \frac{\partial \overline{\tau_{ij}}}{\partial x_j}, \tag{4a}$$

$$\frac{\partial \overline{u_i}}{\partial x_i} = 0, \tag{4b}$$

where the over-bar refers to the filtered fields. Because the NS equations are nonlinear, their convolution contains an additional term not present in the original equation set, corresponding to $\partial_j \overline{\tau_{ij}}$ in equation (4). By straightforward application of the rules of convolution, it can be shown that $\overline{\tau_{ij}}$ has the form

$$\overline{\tau_{ij}} = \overline{u_i u_j} - \overline{u_i}\,\overline{u_j}. \tag{5}$$

and is identified with the residual stress tensor. The value of $\overline{\tau_{ij}}$ is not known *a priori* as it contains the unknown velocity components u_i. In order to close this set of equations, we employ the standard Smagorinsky model for $\overline{\tau_{ij}}$ using

$$\overline{\tau_{ij}} = -2\nu_t \overline{s_{ij}}, \qquad \overline{s_{ij}} = \frac{1}{2}\left(\frac{\partial \overline{u_i}}{\partial x_j} + \frac{\partial \overline{u_j}}{\partial x_i}\right), \tag{6}$$

where ν_t is the so-called eddy viscosity, given by

$$\nu_t(\boldsymbol{x}, t) = C_S^2 \Delta^2 \, |\bar{s}|, \qquad |\bar{s}| = \sqrt{2(\overline{s_{ij}})(\overline{s_{ij}})}, \tag{7}$$

where C_S is the dimensionless Smagorinsky coefficient and Δ is the length scale of the largest unresolved eddy present in the turbulent flow. In this article, we use $C_S = 0.1$ [18] and $\Delta = 2(\Delta x \Delta y \Delta z)^{1/3}$ [12], where $(\Delta x, \Delta y, \Delta z)$ denote the grid spacings in the x, y and z directions, respectively. Finally, we also incorporate a near-wall modelling term of the form

$$\phi_w(z) = \begin{cases} \left\{1 - \exp\left[\left(\frac{-zRe_*}{A}\right)\right]^p\right\}^q & z \leqslant \frac{1}{2} \\ \left\{1 - \exp\left[\frac{-(1-z)Re_*}{A}\right]^p\right\}^q & z \geqslant \frac{1}{2} \end{cases} \tag{8}$$

as an additional pre-factor for the length scale Δ in order to take into account the increased turbulence production close to the walls at $z = 0, 1$. In this article, we take the values $p = q = 1$ and $A = 25$, in keeping with the standard Van Driest components [19]. Thus, the eddy viscosity term becomes $\nu_t = (C_S \Delta \phi_w)^2 |\bar{s}|$. Incorporating the Smagorinsky model as given by equation (6) into the filtered momentum equation (4*a*) yields

$$\frac{\partial \overline{u_i}}{\partial t} + \overline{u_j}\frac{\partial \overline{u_i}}{\partial x_j} = -\frac{\partial \overline{p}}{\partial x_i} + \frac{\partial}{\partial x_j}\left[\left(\frac{1}{Re_*} + \nu_t\right)\left(\frac{\partial \overline{u_i}}{\partial x_j} + \frac{\partial \overline{u_j}}{\partial x_i}\right)\right], \tag{9a}$$

$$\frac{\partial \overline{u_i}}{\partial x_i} = 0, \tag{9b}$$

which are immediately reminiscent of the dimensionless NS equation (2). The key difference is that incorporating the Smagorinsky model effectively introduces a non-constant viscosity term $\nu_T = Re_*^{-1} + \nu_t$ into the equations.

3. S-TPLS solver

The aim of this section is to provide an overview of the numerical algorithms used to solve equation (9) numerically. We have also made available as supplementary material a document that describes in much greater detail several key aspects of the S-TPLS numerical method. This can also be read in conjunction with the available source code (see section 6), such that readers will be able to familiarize themselves with the source code and compile and use it with confidence.

As stated previously, the full research-level version of TPLS solves the NS equations for two interacting fluids. Since we are only interested in single-phase flow in this article, the code used herein is simplified by stripping away the second fluid to create a single-phase NS solver—hence, S-TPLS. However, since a two-phase flow has a non-constant viscosity in the levelset formalism [20], the generic structure of the algorithms in TPLS is ideally suited to handling a non-constant viscosity such as that in equation (9). In presenting the computational methodology below, we omit the overbar over the spatially filtered velocity and pressure fields, since the context of the presentation indicates when we are dealing with an LES simulation.

3.1. Discretization

The domain Ω is discretized in a uniform manner using a finite-volume scheme, resulting in a uniform computational mesh with (N_x, N_y, N_z) grid points in the (x, y, z) directions, and corresponding uniform grid spacings $(\Delta x, \Delta y, \Delta z)$. The approach is similar to a finite-difference method, in the sense that the problem variables are evaluated at discrete points on the mesh. However, the physics of the momentum equation in a small volume or cell surrounding each mesh point is given extra consideration in a finite-volume scheme: the momentum source term is split up into two new terms describing body forces and surface forces. By definition, the surface-force term is the one identifiable with the divergence of a flux—this is then regarded as an integral over the small cell, and is then converted into a sum of surface integrals, using the divergence theorem. These surface integrals are then evaluated as fluxes through the faces of each cell. In this way, the numerical method faithfully represents the underlying physics of such surface forces.

A further unavoidable level of refinement is introduced by way of a marker-and-cell (MAC) grid [21] (figure 2). The values for p and ν_T are stored at the cell centres while velocity components are stored at the cell faces. This approach stabilizes the code numerically against the checkerboard instability [21]. Additionally, the use of a MAC grid allows spatial velocity derivatives to be approximated numerically as centred differences taken between two cell faces, accurately taking into account the momentum flux between cells.

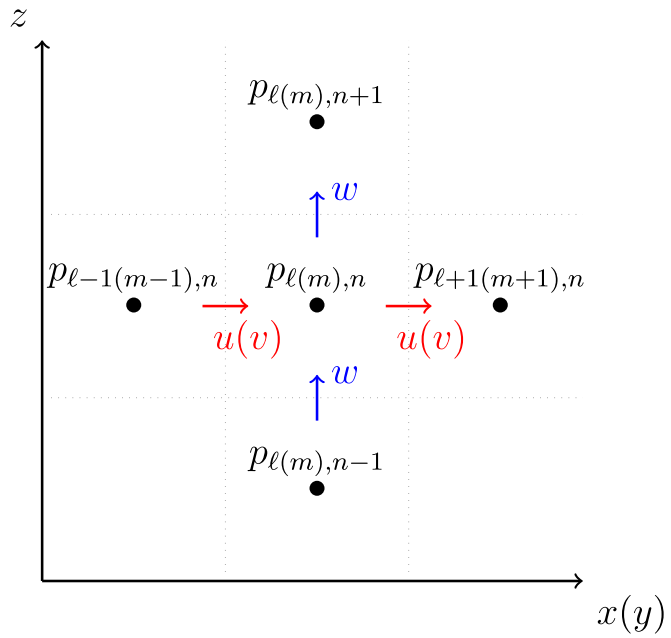

Figure 2. *S*chematic of a two-dimensional MAC grid, in the *xz* or equivalently *yz* plane, indicating the location of pressure and velocity values. One can immediately envision the extension of the scheme to three-dimensions.

Concerning the temporal discretization of the code, the momentum equation (9*a*) is solved using the so-called projection method [22]. This amounts to an operator-splitting technique, whereby the passage from one timestep to the next is broken up into two half-steps. An advantage of this approach is that the computations of the updated velocity and pressure fields become decoupled. In this method, the pressure term is first omitted from the momentum equation which is then solved for at an intermediate half-step $\boldsymbol{u}^*$ i.e.

$$\frac{\boldsymbol{u}^* - \boldsymbol{u}^n}{\Delta t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} = \nabla \cdot (\nu_T (\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T)), \tag{10}$$

where $\boldsymbol{u}^n$ is the velocity field at the nth time step, Δt is the time step, and the total viscosity ν_T is a time- and space-dependent variable. The viscous derivative is split into terms which appear more convective and others which appear more diffusive in nature. For example, by expanding the term on the right hand side of equation (10), one obtains terms such as (again for $j = 1, 2, 3$)

$$\frac{\partial}{\partial x_i}\left(\nu_T \frac{\partial u_i}{\partial x_j}\right), \tag{11}$$

which we recognise to be convective in nature, and terms which appear more diffusive

$$\frac{\partial}{\partial x_i}\left(\nu_T \frac{\partial u_j}{\partial x_i}\right). \tag{12}$$

The first of these terms (i.e. equation (11)) is discretized in time using a third-order Adams–Bashforth scheme [23]. The same temporal discretization scheme is used for the convective derivative $\boldsymbol{u} \cdot \nabla \boldsymbol{u}$. A Crank–Nicolson scheme [24] is used for the more diffusion-like terms in

equation (12). Equation (10) is then solved numerically for $\boldsymbol{u}^*$, which is then used to obtain the velocity field at the next time step $\boldsymbol{u}^{n+1}$ by reintroducing the pressure term and making the following correction to $\boldsymbol{u}^*$

$$\frac{\boldsymbol{u}^{n+1} - \boldsymbol{u}^*}{\Delta t} = -\nabla p^{n+1}. \tag{13}$$

Taking the divergence of both sides and using the incompressibility requirement on the field $\boldsymbol{u}^{n+1}$ i.e. $\nabla \cdot \boldsymbol{u}^{n+1} = 0$ one obtains

$$\nabla^2 p^{n+1} = \left(\frac{\nabla \cdot \boldsymbol{u}^*}{\Delta t}\right), \tag{14}$$

i.e. Poisson's equation. Once p^{n+1} is determined, this allows us to solve for $\boldsymbol{u}^{n+1}$ via equation (13).

3.2. Error analysis, code stability, and convergence

We briefly discuss the error associated with the chosen numerical method, and explain how those errors can be minimized. Addressing these issues is best practice in computational science and makes the presented results robust. Centred differences are used throughout the code in evaluating spatial derivatives. The centred differences are implemented on the MAC grid. Thus, the error associated with approximating $\partial\phi/\partial x$ by a finite difference is $O(\Delta x^2)$, where ϕ is a generic field variable. Concerning the convective derivative $\boldsymbol{u} \cdot \nabla \boldsymbol{u}$, a more accurate treatment would involve upwinding and possibly accounting for local sharp changes in the velocity field (for example, using a (weighted) ENO scheme [25, 26], as in [20]). However, the centred differences are shown in the simulation results to be adequate for our purposes.

The temporal discretization involves both a Crank–Nicholson discretization for the diffusion and a third-order Adams–Bashforth step for the convective derivative. The schemes introduce an error over the course of the entire simulation that is $O(\Delta t^n)$, with $n = 2$ for the Crank–Nicholson part [23] and $n = 4$ for the Adams–Bashforth part [27]. The explicit treatment of the convective derivative in the temporal discretization introduces a CFL stability constraint on the code [23]. A conservative rule of thumb to maintain stability in the simulations is found to be

$$U_{\max}\Delta t \leqslant 0.1\Delta x, \tag{15}$$

where $U_{\max}$ is the maximum streamwise velocity. Finally, the simulations are checked for convergence of the solutions with respect to the grid spacings Δx, Δy, Δz, and the stepsize Δt. The turbulence simulation requires additional care in selecting converged values of the grid spacings, as the grid spacings must be small so that the LES captures enough of the energy-containing motions so as to produce accurate statistics for the mean flow.

3.3. Domain decomposition and output data

The presented code is implemented in Fortran 90 with MPI parallelization using a domain decomposition [28]. This means that the code splits the computational domain into several smaller sub-domains at runtime (as in figure 3). Thus, the three-dimensional arrays that store the physical variables are split into smaller arrays corresponding to the different sub-domains. The numerical computations on each of the smaller arrays are done in parallel on separate CPU cores. In the course of these computations, each smaller array needs information from neighbours—this is obtained using the MPI library which passes messages between the

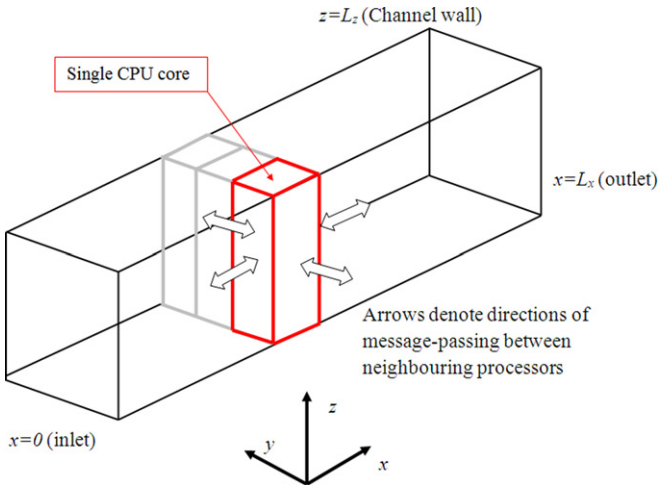

Figure 3. Schematic diagram showing the domain decomposition for the parallel code. Geometrically, each sub-domain is an elongated box spanning the entire wall-normal direction. The Navier–Stokes equations are solved on each sub-domain on a particular CPU core. Information is passed between the sub-domains using the MPI library.

different CPU cores, thereby exchanging the necessary data between the different computational sub-domains. It should be emphasized that all of these tasks of parallelizing the code are performed by calls to the MPI library, and are already implemented in S-TPLS. Moreover, many of the MPI aspects of S-TPLS appear only as subroutines requiring no user interaction, leaving the user free to focus on the computational algorithms and the physics.

The sub-domains associated with the domain decomposition (figure 3) span the entire wall-normal direction, such that the decomposition is two-dimensional. The reason for this is that the wall-normal boundary conditions are difficult to implement in practice on a MAC grid, meaning that parallelization in this direction is undesirable. A second advantage of this approach is that it enables the user to introduce different physical effects into the code concerning the interplay between the fluid and the solid wall (e.g. surface roughness), without interacting with the MPI aspects of the code.

The code periodically dumps the pressure, velocities, and viscosity to a series of files for postprocessing and visualization. In S-TPLS, this is done in a simple way, whereby all data is gathered to a single CPU processor and written to a single series of files. The files are ASCII-formatted and configured for visualization using proprietary software (Tecplot). However, because of the simplicity of the file structure the data can easily be accessed for visualization and postprocessing with other software. A drawback of this approach is that the data output is performed on a single CPU processor, which is a bottleneck and limits the code's scalability. For this reason, the full research-level version of TPLS uses parallel I/O with NetCDF [10].

3.4. Scheduled relaxation Jacobi method

The research-level version of the TPLS code uses calls from the Fortran code to the PETSc linear-algebra library to solve the Poisson equation for the pressure-correction step, which leads to a highly efficient parallelized pressure solver [1]. However, in order to provide students with a simple, robust and highly portable code in the present version we have implemented instead the recently discovered SRJ scheme [29]. In this way, the simplified

code contains no calls to external libraries (other than standard MPI), and can be compiled on a desktop, a cluster, or a supercomputer using a standard Fortran 90 compiler with an MPI wrapper. The basic idea behind SRJ is explained in what follows.

Iterative methods to solve Poisson's equation numerically are often discussed in undergraduate computational science classes [24]. The Jacobi method is the most primitive of these methods; a more refined version is the weighted Jacobi method with non-negative weight ω [29, 30]. However, use of weighted Jacobi method does not ensure convergence for all wavenumbers [29]. Hence this method is usually abandoned in favour of the Gauss–Seidel method with successive over-relaxation (SOR) which (for Poisson's equation) converges for $\omega \leqslant 2$. The SRJ method, on the other hand, proposes the use of a combination of over-relaxation parameters ($\omega > 1$) and under-relaxation parameters ($\omega < 1$) applied to the weighted Jacobi method. Essentially, a fixed number M iterations of the Jacobi-SOR method are carried out, each of which has a prescribed relaxation parameter ω_k for k ranging from 1 to P, $P \geqslant 2$. These values are not necessarily unique, with each ω_k repeated q_k times respectively, and the M-iteration cycle is then repeated until convergence. This method has been motivated by the fact that over-relaxation of the Jacobi method reduces the low wavenumber error while under-relaxation tends to reduce the high-wavenumber error [29].

In the context of this report, the SRJ method for the three-dimensional Poisson problem (14) has been incorporated into S-TPLS. For given values of (N_x, N_y, N_z) on a unit domain, one can choose an appropriate set of relaxation parameters w_k based upon a characteristic number of grid points N. The relaxation schedule (i.e. the order in which the w_k should appear in the M-iteration) is then obtained courtesy of the Matlab script provided in [29].

3.5. Validation

Given the sheer size of the S-TPLS code, its combination of many algorithms, and its incorporation of the MPI parallel programming methodology, it is essential to be able to validate its accuracy and fidelity to the underlying equations of motion, before running expensive simulations and making predictions concerning the properties of the flow under consideration. Linear stability analysis provides a rigorous test case with which to validate S-TPLS—at least prior to incorporation of the LES technique. The reason is that the linear stability analysis of a steady base state yields a non-trivial quasi-analytical temporally evolving perturbed state, which can be compared to a similar state emanating from a direct numerical simulation. In the context of the present model problem, the linear stability theory is known as an Orr–Sommerfeld analysis [31]. A comparison along these lines between S-TPLS and the semi-analytic Orr–Sommerfeld theory is presented in the appendix, wherein excellent agreement between the two approaches is obtained, confirming the accuracy of S-TPLS in simulating temporally evolving channel flows.

4. Performance analysis

Before embarking on a large-scale LES of turbulent channel flow, it is of interest to know the most efficient way in which to deploy the parallelism of S-TPLS with both the SRJ pressure solver and the LES technique incorporated into the code. In this section, we therefore investigate the strong scaling behaviour of the code i.e. how its performance varies with the number of cores used for a fixed problem size. This information then enables us to choose the number of CPU cores so as to maximize the efficiency of the code. The information is also of more general interest, as it enables one to quantify the trade-off between execution time and computational resources. Moreover, the ability to conduct a rigorous performance analysis is

an important skill for researchers to possess, as such preparatory work is often required before a code can be used in national supercomputing centres [32]. Thus, this section presents standard practice in this skill to interested students.

Using an appropriate domain and grid resolution, the S-TPLS solver is run for a total of 1000 iterations using N_p CPU cores. Omitting time associated with initialization and periodic file output, the execution time of the code is recorded as T_p and the number of processes used varied over the range $N_p = 24$–1008. The value $N_p = 24$ is a result of the architecture of the machine used for the calculation (Fionn, ICHEC): each compute node on Fionn consists of 24 cores, and an efficient use of the resources requires that an integer number of nodes be reserved for each simulation. The analysis parameters are outlined in table 1. One notes the use of a 'long' domain and a higher resolution in the wall-normal direction to that used in the streamwise and spanwise directions. This is due to the fact that the most vigorous turbulence production occurs close to the walls [14], thus requiring a higher grid resolution.

Having obtained the execution times (given in table 2), one can calculate associated quantities such as the speedup S_p and parallel efficiency E_p which are given by

$$S_p = \frac{T_{24}}{T_p}, \quad E_p = \frac{24T_{24}}{N_p T_p}. \tag{16}$$

Following standard practice [32], these quantities are plotted in figure 4. Evidently, one can observe from figure 4(a) that super-linear speedup is achieved for process counts $N_p \leqslant 336$, with sub-linear speedup beyond this point. Moreover, one can observe that $E_p < 1$ for values of $N_p \gtrsim 385$. Superlinear speedup is generally associated with an efficient use of the cache by the code as memory required by the sub-problem on a particular CPU core is reduced under the domain decomposition. In contrast, sublinear speedup at higher process numbers is associated with communication overheads, which we investigate below.

To understand the communication overhead associated with running the code in parallel, the code was profiled using the Allinea MAP tool provided on the ICHEC system. Profiling the code allows one to investigate how much time is spent on each individual part of the code. In parallel computing, profiling is extremely useful as it identifies the most time-intensive parts of the code which can then be refined and optimized, so as to achieve increased efficiency. Using the same parameters as outlined in table 1, two sample profiles are constructed (table 3). One can see that the solution of the Poisson equation for the pressure is by far the most intensive aspect of the 'maths' part of the code, far outweighing the time spent computing the velocity field. This is understandable given that one must use an iterative process for a very weakly diagonally dominant system (i.e. Poisson's equation) in order to solve for the pressure. Communication overheads (MPI send-receives) are also a dominant feature, with over half the execution time (55.7%) spent on communication between processors for $N_p = 216$. These overheads will continue to increase as N_p is increased, leading to a decrease in E_p as evident from figure 4(b).

In summary, the efficiency of the parallel code is much reduced at large CPU core counts. This can be viewed as the penalty for using S-TPLS, as the parallel efficiency of the research-level code which uses the PETSc and NetCDF libraries has been measured to be 0.9 at $N_p > 2000$ [1]. For the present pedagogical applications, this loss of efficiency is not severe, and only manifests itself at large CPU core counts (which would anyway correspond to research-level simulations). Also, the loss of efficiency is compensated by the increase in the code simplicity and portability.

Table 1. Parameters used for the performance analysis of the S-TPLS code incorporating both the SRJ solver and LES technique.

Re_*	N_x	N_y	N_z	N_T	Δx	Δy	Δz	Δt
360	288	126	120	4 354 560	$(36)^{-1}$	$(36)^{-1}$	$(120)^{-1}$	5×10^{-5}

Table 2. Execution times T_p. Note that the choice of multiples of 24 for the number of processes N_p was due to the architecture of the machine used (Fionn, ICHEC), and is not a requirement of the code itself.

Number of processors N_p	Execution time T_p (seconds)
24	3669.86
$3 \times 24 = 72$	557.45
$7 \times 24 = 168$	342.76
$14 \times 24 = 336$	248.26
$28 \times 24 = 672$	193.94
$42 \times 24 = 1008$	186.91

Table 3. Profile illustrating the most time intensive parts of the single phase SRJ LES solver. Note that MPI send-receives represents time spent by processors communicating with one another.

	Percentage of total time			
Number of processors N_p	Pressure calculation	MPI send-receives	Velocity calculation	Other
72	51.3	39.6	8.7	0.4
216	37.6	55.7	6.2	0.6

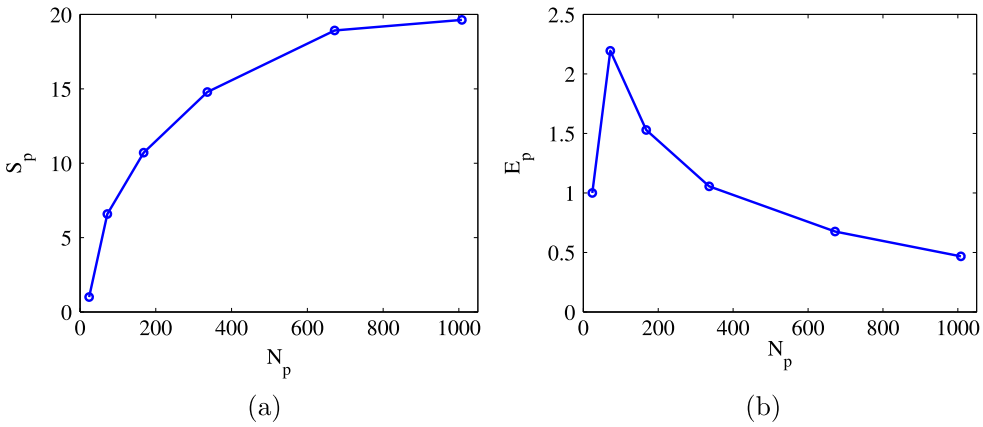

Figure 4. (a) Speedup curve S_p and (b) parallel efficiency curve E_p.

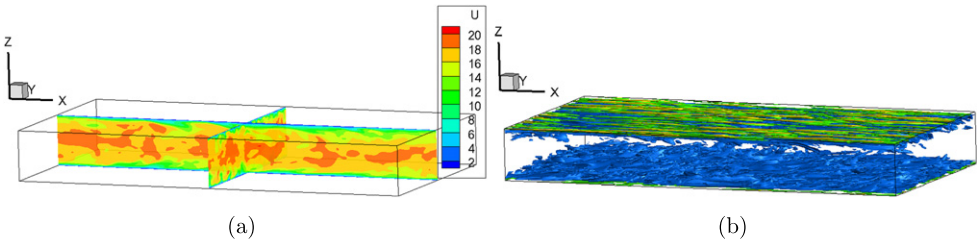

Figure 5. Snapshot of instantaneous motion, $t = 21.6$. Panel (a): instantaneous streamwise velocity showing the maximum streamwise velocity near the centreline; panel (b): isosurfaces of instantaneous vorticity magnitude showing intense turbulence generation near the walls. The isosurfaces in green closest to the walls correspond to $|\nabla \times \boldsymbol{u}| = 300$; the isosurfaces in blue with contributions further from the walls correspond to $|\nabla \times \boldsymbol{u}| = 100$.

5. Results of the LES

In the present section we study pressure-driven channel-flow turbulence, using the large-eddy concept developed in sections 1 and 2. Rather than introducing original research, the aim here is to showcase the ability of S-TPLS to carry out intensive simulations in an efficient manner, and the computational results are compared with existing standard results from the literature. In presenting the below results, we omit the overbar over the spatially filtered velocity and pressure fields, since it is clear that we are dealing with an LES simulation.

We focus on fully developed turbulence, rather than the laminar-turbulent transition. Therefore, the simulation is seeded with a turbulent-like initial condition, so that the simulation settles down rapidly to a fully-developed state, for which the statistics of the flow can be gathered. Specifically, we take [33, 34]

$$\boldsymbol{u}(\boldsymbol{x},\, t=0) = aU_0(z) + \nabla \times [f(z)\boldsymbol{A}(\boldsymbol{x})], \qquad p(\boldsymbol{x},\, t=0) = \frac{\mathrm{d}p}{\mathrm{d}x}x, \tag{17a}$$

where $U_0(z)$ is the base-state Poiseuille flow given by equation (A.2), the function $f(z)$ is given by equation (A.8*b*), and $\boldsymbol{A}(\boldsymbol{x})$ is a collection of random Fourier modes given by

$$A_i(\boldsymbol{x}) = \sum_{k_x=1}^{C_{k_x}} \sum_{k_y=1}^{C_{k_y}} \sum_{k_z=1}^{C_{k_z}} [S_i(k_x,\, k_y,\, k_z)\sin(\boldsymbol{k}\cdot\boldsymbol{x}) + C_i(k_x,\, k_y,\, k_z)\cos(\boldsymbol{k}\cdot\boldsymbol{x})], \tag{17b}$$

where we have a sum over the wavenumber $\boldsymbol{k}$:

$$\boldsymbol{k} = \left(\frac{2\pi}{L_x}k_x,\, \frac{2\pi}{L_y}k_y,\, \frac{2\pi}{L_z}k_z\right), \tag{17c}$$

and the constants $S_i(k_x,\, k_y,\, k_z)$ and $C_i(k_x,\, k_y,\, k_z)$ are random numbers between 0 and 1. The term $\nabla \times [f(z)\boldsymbol{A}]$ represents a manifestly incompressible fluctuation, while $aU_0(z)$ represents a crude approximation to the turbulent mean flow, which is reduced and 'flattened' with respect to its laminar counterpart [14]—we therefore take $a = 1/3$.

Based on the above, an LES was performed using the parameters in table 4, using five Fourier modes in each spatial direction for the initial condition in equation (17*b*). Snapshot results are shown in figure 5. The snapshots reveal the expected qualitative features of turbulent channel flow, in particular the streamwise maximum velocity near the channel centreline and the intense turbulence generation near the walls, as evidenced by the large

Table 4. Simulation parameters used for LES. The number of CPU cores used is $N_p = 336$ on Fionn, chosen to correspond to a regime wherein the parallel efficiency of the code is close to 100%. Also, the size of the grid in the x- and y-directions is chosen with respect to a reference paper [16] so that the turbulent structures do not interact with each other through the periodic boundary conditions.

Re_*	N_x	N_y	N_z	L_x	L_y	L_z	Δt
360	240	140	120	8	4	1	10^{-4}

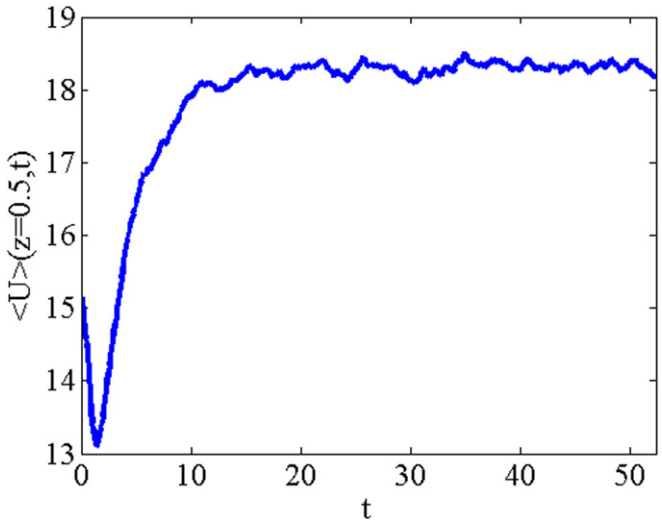

Figure 6. Time dependence of the mean centreline velocity showing the convergence to a statistically steady state.

vorticity magnitude in figure 5. The code was run for 53 dimensionless time units, and a statistically steady state was attained after 20 time units. The onset of the statistically steady state was determined by inspection of the average channel centreline velocity $L_x^{-1}L_y^{-1}\int_0^{L_x} \mathrm{d}x \int_0^{L_y} \mathrm{d}y\, u(x, y, z = 0.5, t)$: after some transience this fluctuates around a steady value (figure 6).

The instantaneous velocity fields (such as that in figure 5(a)) are averaged over time $t \geqslant 20$ and over the x- and y-directions to produce space–time average quantities: for a property $\psi(x, y, z, t)$ we define the spacetime average as

$$\langle\psi\rangle(z) = \frac{1}{t_2 - t_1}\frac{1}{L_x L_y}\int_{t_1}^{t_2} \mathrm{d}t \int_0^{L_x} \mathrm{d}x \int_0^{L_y} \mathrm{d}y\, \psi(x, y, z, t),$$

with $t_1 = 20$ and $t_2 = 53$. The structure of the mean flow $\langle u\rangle(z)$ is obtained in this way and the results are presented in figure 7. The characteristic 'flattenend' profile is shown in figure 7(a). The same profile is shown in wall units on a semi-logarithmic scale in figure 7(b). This profile is consistent with the 'universal' description of wall-bounded turbulence [14, 15], with $\langle u\rangle \sim z^+$ near the wall, and a 'log layer' $\langle u\rangle = (1/0.4)\log z^+ + 5.5$ in an intermediate zone between the wall and the channel centreline [14, 15]. Here $z^+ = zRe_*$ denotes the wall-normal coordinate expressed in wall units.

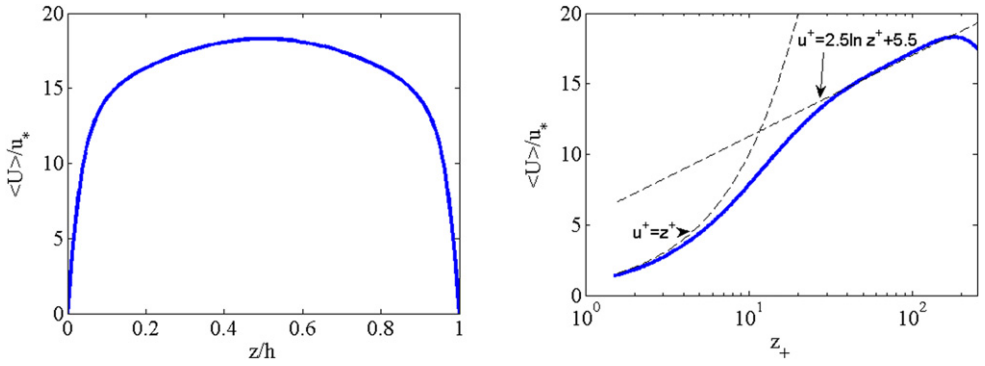

Figure 7. The mean velocity profile for fully developed flow at $Re_* = 360$. The centreline velocity agrees with the correlation $U_{\max}/u_* = (1/0.4)\log Re_* + 3.47$ from the literature [14]. (b) The mean velocity profile, showing the viscous sublayer $\langle u\rangle \sim z^+$ and $\langle u\rangle = (1/0.4)\log z^+ + 5.5$: the logarithmic layer appears in the region $50 < z^+ < 150$. Here $z^+ = zRe_*$ denotes the wall-normal coordinate in wall units.

We also compare these results with our knowledge of the 'bulk' properties of the flow, i.e. properties obtained by averaging over all spatial dimensions. We introduce

$$U_{\text{mean}} = \frac{1}{L_z}\int_0^{L_z} \mathrm{d}z\ \langle u\rangle.$$

Based on figure 7, this is computed to be $U_{\text{mean}} = 15.39$, hence $Re_{\text{mean}} = U_{\text{mean}}H/\nu = 5542$. The centreline velocity derived from the same figure is 18.30, in agreement with the known correlation [14] $U_{\max} \approx (1/0.4)\log Re_* + 3.47 = 18.18$. Thus, the ratio $U_{\max}/U_{\text{mean}} = 1.19$ is obtained, which is close to the value of 1.16 obtained in the direct numerical simulation by Kim *et al* [16]. The skin-friction coefficient, $C_{\text{f}} = 2u_*^2/U_{\text{mean}}^2 = 8.44 \times 10^{-3}$($8.18 \times 10^{-3}$ in the direct numerical simulation by Kim *et al*). This value also fits the correlation of Dean [35], who suggested a value of $C_{\text{f}} = 0.073Re_{\text{mean}}^{-0.25}$, corresponding to $0.073 \times 5542^{-0.25} = 8.46 \times 10^{-3}$ in the present simulation.

We examine the following two-point correlation functions

$$R_{ij}(x, y, z) = \frac{1}{t_2 - t_1}\frac{1}{L_xL_y}\int_{t_1}^{t_2}\mathrm{d}t\int_0^{L_x}\mathrm{d}x'\int_0^{L_y}\mathrm{d}y' u_i(x + x', y + y', z)u_j(x', y', z) - \langle u_i\rangle(z)\langle u_j\rangle(z);$$

in particular we investigate the streamwise correlations $R_{ij}(x) \coloneqq R_{ij}(x, 0, 1/2)$, the results of which are shown in figure 8(a). The two-point correlations vanish at large separations, which confirms that the choice of $L_x = 8$ is adequate for the simulation. To further understand the distribution of the turbulent kinetic energy among the different length scales we introduce the Fourier transform of $R_{ij}(x, y, z)$ in a plane parallel to the walls at location z, defined as follows:

$$\hat{R}_{ij}(k_x, k_y, z) = \iint_{[0,L_x]\times[0,L_y]} \mathrm{d}x\mathrm{d}y\ \mathrm{e}^{-\mathrm{i}k_xx - \mathrm{i}k_yy}R_{ij}(x, y, z). \tag{18}$$

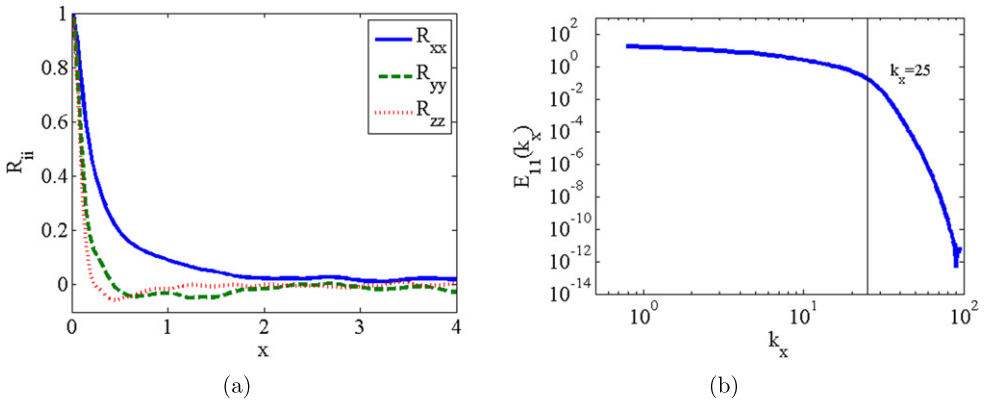

Figure 8. (a) Streamwise two-point correlation functions and (b) Fourier transform of the streamwise correlation R_{xx}. The onset of the dissipation range is shown in (b).

Equation (18) is crucial, as it enables one to connect the simulation data to Kolmogorov scaling theory [14]. In particular, we examine $E_{11}(k_x, z) := |\hat{R}_{ij}(k_x, 0, z)|$ at the channel midpoint $z = 1/2$ in figure 8(b). The Kolmogorov scaling law $E_{11}(k_x) \sim k_x^{-5/3}$ valid asymptotically at high Reynolds numbers is not in evidence but this is not surprising, as the present simulation is performed at $Re_* = 360$, which is at the low end for a turbulent flow. Indeed, the results are qualitatively very similar to the observed and model results for low-Reynolds number turbulence, in particular the data and model curves for the universal one-dimensional longitudinal velocity spectra given in [14] (chapter 6 therein). The spectrum falls off in exponential-type manner for large wavenumbers, corresponding to the dissipation range. More precisely, the onset of the dissipation range is expected for $k\eta \approx 0.3$, where η is the Kolmogorov scale. The Kolmogorov lengthscale is calculated from $\eta = (\nu^3/\epsilon)^{1/4}$, where $\epsilon = Re_*^{-1}\langle s_{ij}s_{ij}\rangle$ is the dissipation rate of turbulent kinetic energy and where s_{ij} is the fluctuating component of the rate-of-strain tensor. For the present simulation, the Kolmogorov lengthscale is calculated to be approximately two wall units, hence $\eta = 2/Re_*$ in non-dimensional terms In contrast, in figure 8(b) one can see that the onset of the dissipation range occurs at $k\eta \approx 2 \times 25/Re_* \approx 0.1$. Yet again, this is consistent with our knowledge of LES: by design, the Smagorinsky model introduces extra dissipation in the small scales (yet starting at scales larger than the Kolmogorov lengthscale), so the early onset of the dissipation range in figure 8(a) is expected. In conclusion, the present study confirms that LES is highly effective in capturing the bulk statistics quantitatively very well (e.g. the spatial structure of the mean flow, the Reynolds stress, and the turbulent kinetic energy in the wall-normal direction), yet only captures the fine-scale statistics in a qualitative manner, albeit that the spectra emanating from the LES can be put on a very firm theoretical footing [14].

6. Conclusions and didactic considerations

Summarizing, the modified and simplified S-TPLS solver introduced in this work performs well in simulating single-phase pressure-driven channel flow; in particular, the LES model accurately describes turbulence phenomena in a channel flow, as evidenced by a rigorous comparison between our own simulation results and the standard results from the literature.

We also discuss herein how our computational methodology fits inside the broader context of CFD education. While there are several commercial CFD solvers taught in current undergraduate and graduate science curricula the world over, the emphasis is more on a 'black-box' approach with focus on handling the software for standard problems including turbulent flow in a channel/pipe. Typically, these standard CFD courses are delivered predominantly as electives in the graduating years, with students having taken fundamental courses in fluid mechanics courses in the earlier years. However, more often, there is a mismatch in the learning objectives and delivery aspects for both these courses—with a CFD course appearing to adopt a more black-box type of approach—leading to less emphasis on the fundamental physics governing the flows. Moreover, the common commercial software being used for these courses provide only a graphical user interface (GUI), with little or no access to the solvers being used. While this is rightly so to protect IP and also ensure stability of the code as it is less amenable to tinkering, the students only have experience in treating a flow problem using a GUI that is highly specific to the software available in their curricula. This is particularly true for complex flow phenomena such as turbulence where a RANS-type model with specified eddy viscosity closures are predominantly promoted as the method of choice in commercial flow solvers—with little or no means of introducing even rudimentary levels of customization. Also, a typical turbulent flow example within vendor-prescribed tutorials (in commercial codes) would be based on a 'steady-state two-dimensional or axi-symmetric' solution—giving a completely incorrect impression to the students that turbulence is steady and two-dimensional. While it is incumbent on the instructor to correct this impression—examples of such fictitious flows are too ubiquitous (in almost every commercial solver and even old peer-reviewed articles) for students to ignore them. Of course, this is gradually changing with availability of LES methods in some solvers which will only work with transient approaches—but again only with rigidly specified models for Reynolds stresses to account for sub-grid turbulence. A further problem is the terminology used in commercial solvers, where they prescribe a DNS approach as a 'laminar' model. Undergraduate students are then exposed to a risky solecism—believing that a DNS approach is 'unsuitable' for turbulent flows. While many university departments have facilities for local computing clusters, the standard parallelization schemes in commercial CFD solvers renders their efficiency sub-optimal. This means that the undergraduate students are only exposed to approximate 'simple' single-core problems for 'laminar' (low Re) or 'steady-state 2D/axi-symmetric RANS-type turbulent' (high Re) flows and never get hands-on experience with high-performance parallelized computing for transient real-life flows (which is mostly exclusive to senior graduate students or post-doctoral researchers).

S-TPLS is a first step in addressing these shortcomings in CFD education. Crucially, it is free open-source software and can be downloaded from a repository: https://sourceforge.net/p/tpls/code/HEAD/tree/trunk/s-tpls/. Thus, the code is readily available to students for immediate use, and its open-source nature means that students can familiarize themselves with the standard algorithms implemented in the code. A further advantage is that the code is fully parallelized in a simple but robust way, meaning that the code can be implemented on a wide variety of platforms, from desktops to compute clusters, even up to implementation on supercomputers. Since the code uses only standard MPI and Fortran, it is highly portable, thereby further enabling easy access by students. Based on our own experience, students are able to run the code on a desktop machine in a timely fashion (albeit with resolution of only the largest eddies) and hence gather some rough statistics to characterize the turbulence. A further final advantage of open-source software is the ability of users to freely modify the source code. Therefore, the present implementation of TPLS can be used not only for study

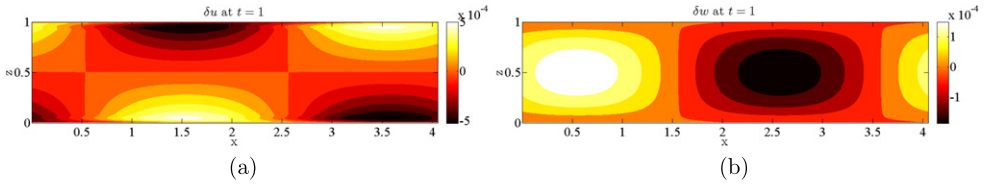

Figure A1. Plots (a) and (b) illustrate the perturbation velocities at $t = 1$ as determined from a numerical simulation using S-TPLS. The initial conditions are those given in equation (A.8).

and instruction but also as a starting-point for research into temporally evolving three-dimensional flows.

Acknowledgments

LÓN and JF acknowledge the DJEI/DES/SFI/HEA Irish Centre for High-End Computing (ICHEC) for the provision of computational facilities and support. LÓN and JF also acknowledge Hendrik Hoffmann's maintenance of the Orr computer cluster in the School of Mathematics and Statistics in UCD and support for users of the same.

Appendix. Validation

Orr–Sommerfeld theory provides a known semi-analytical solution of the channel-flow problem, at least prior to the incorporation of the LES technique into the model equations. Thus, in this appendix the Orr–Sommerfeld theory is introduced and then used as a stringent test to demonstrate the validity of the S-TPLS code. It is justifiable to validate the code prior to the incorporation of the LES technique, as the latter is a simple 'add-on' to the basic hydrodynamics modules in the code. Thus, the requirement that the pre-LES version of the code should agree precisely with Orr–Sommerfeld theory is a necessary condition for the code's correctness. Subsequent to the incorporation of the LES technique, validation can be done *a posteriori*, e.g. by examining the properties of the numerically-computed mean flow and comparing to previous numerical and experimental works.

A.1. Orr–Sommerfeld analysis

We consider the laminar flow analogue to the model problem given in section 2. In this case, the problem reduces to a two-dimensional one i.e. in the xz plane, with the flow being both uni-directional and steady. As such, enforcing no-slip boundary conditions on the walls leads to the standard Poiseuille velocity profile

$$U_0(z) = \frac{h^2}{2\mu}\left|\frac{\mathrm{d}p}{\mathrm{d}x}\right|\frac{z}{h}\left(1 - \frac{z}{h}\right), \tag{A.1}$$

where we use the fact that the pressure gradient is inherently negative and $L_z = h$ is the channel height. Using the same non-dimensional formulation as in section 2, this can be written in non-dimensional form as

$$U_0(z) = Re_* z(1 - z). \tag{A.2}$$

The idea behind the Orr–Sommerfeld analysis is to introduce a small perturbation to this base-state profile (denoted using subscripts 0) of the form

$$(u, w, p) = (U_0 + \delta u, \delta w, p_0 + \delta p), \tag{A.3}$$

where δu, δw, and δp are the perturbations that depend both on space and time. We now perform a linear stability analysis by substituting this velocity profile into the non-dimensional NS equation (2) and ignoring terms which are quadratic in the perturbation velocities. Linearising about the base flow yields the following set of linear disturbance equations:

$$\frac{\partial(\delta u)}{\partial t} + U_0\frac{\partial(\delta u)}{\delta x} + \delta w\frac{\partial U_0}{\partial z} = -\frac{\partial(\delta p)}{\partial x} + \frac{1}{Re_*}\nabla^2\delta u, \tag{A.4a}$$

$$\frac{\partial(\delta w)}{\partial t} + U_0\frac{\partial(\delta w)}{\partial x} = -\frac{\partial(\delta p)}{\partial z} + \frac{1}{Re_*}\nabla^2\delta w, \tag{A.4b}$$

$$\frac{\partial(\delta u)}{\partial x} + \frac{\partial(\delta w)}{\partial z} = 0. \tag{A.4c}$$

As the flow is two-dimensional we can relate the perturbation velocities to the streamfunction $\Psi(x, z, t)$ via

$$\delta u = \frac{\partial\Psi}{\partial z}, \quad \delta w = -\frac{\partial\Psi}{\partial x}; \tag{A.5}$$

moreover, we can make a normal-mode decomposition

$$\Psi = \exp[\mathrm{i}\alpha(x - ct)]\psi(z), \tag{A.6}$$

where $\alpha \in \mathbb{R}$ and $c \in \mathbb{C}$ are the wavenumber and speed of the disturbance, respectively. Such a decomposition is justified, as the general solution of the linear disturbance equation (A.4) can be written as a superposition of plane-wave solutions ('normal modes'); each plane-wave solution has a streamfunction of the form (A.6) which also satisfies the linear disturbance equations. Based on this normal-mode decomposition, after a number of steps, one arrives at the Orr–Sommerfeld equation

$$\mathrm{i}\alpha(U_0 - c)\left(\frac{\partial^2}{\partial z^2} - \alpha^2\right)\psi - \mathrm{i}\alpha U_0''\psi = \frac{1}{Re_*}\left(\frac{\partial^2}{\partial z^2} - \alpha^2\right)^2\psi \tag{A.7}$$

which, supplemented with the no-slip boundary conditions $\psi(z) = \psi'(z)$ for $z = 0, 1$, presents an eigenvalue problem for c. For our purposes, it is important to note that since $\Psi \propto \exp[\mathrm{i}\alpha(x - ct)] = \mathrm{e}^{\mathrm{i}\alpha x}\mathrm{e}^{\lambda t}$, where $\lambda = -\mathrm{i}\alpha c$, the perturbation velocities will grow exponentially in time if $\Re(\lambda) > 0$ (note: since $\lambda = -\mathrm{i}\alpha c$, exponential growth with $\Re(\lambda) > 0$ corresponds to $\Im(c) > 0$ also). Moreover, the perturbation velocities also undergo an oscillation with period $T = 2\pi/\omega = 2\pi/|\Im(\lambda)|$.

The foregoing description pertains to all Reynolds numbers, yet a more precise classification of the solution to the eigenvalue problem (A.7) is provided in the context of hydrodynamic instability by introducing the so-called critical Reynolds number Re_{*c}:

- For $Re_* > Re_{*c}$ one can find a range of unstable wavenumbers for which $\Re(\lambda) > 0$ and the perturbations grow and contaminate the base state. This is referred to in the literature as a *supercritical* case [36].
- For $Re_* < Re_{*c}$ we have $\Re(\lambda) < 0$ for all wavenumbers, hence all perturbations of sufficiently small initial amplitude die out as $t \to \infty$. This is referred to in the literature as a *subcritical* case [36].

Table A1. Simulation parameters used for Orr–Sommerfeld analysis of the S-TPLS code. As S-TPLS is fundamentally a three-dimensional code, the choice $N_y = 3$ (i.e. instead of $N_y = 1$) was to ensure code reliability (especially with respect to the MPI implementation). Consequently, the produced velocity and pressure fields are strongly two-dimensional in nature, corresponding to the theoretical scenario assumed in equation (A.4).

Re_*	N_x	N_y	N_z	Δx	Δy	Δz	Δt	$\Re(\lambda)$	$\Im(\lambda)$	T
500	1200	3	300	$(300)^{-1}$	$(300)^{-1}$	$(300)^{-1}$	10^{-5}	1.8056	−33.873	0.185

The critical Reynolds number for Poiseuille flow was first obtained in [37] and is known to be $Re_{*_c}A \approx 214.9$. (Note that Re_{*_c} is given here in terms of the friction velocity and the channel height, yet in [37] the critical Reynolds number is given in terms of the mean velocity and the channel half-height—under this rescaling the critical Reynolds number for Poiseuille flow assumes the approximate value 5772.)

A.2. Simulation results

The S-TPLS code is configured in order to perform an Orr–Sommerfeld analysis. Simulation parameters used and predicted values for $\Re(\lambda)$ and T are given in table A1 . In particular, the Reynolds number is taken to be $Re_* = 500$, and the wavenumber is taken to be $\alpha = 2\pi/4$, corresponding to a supercritical case with $\Re(\lambda) > 0$. The eigenvalue analysis is conducted using an in-house Orr–Sommerfeld solver which has been validated carefully against the results of Orszag [37]. This is compared with a simulation in a channel of length $L_x = 4$, chosen so as to correspond to the wavelength of the unstable mode.

The initial condition needed to trigger the numerical instability is introduced via an initial incompressible perturbation velocity field added to the base-state profile given by equation (A.2). The perturbation velocity field is given here:

$$\delta u = -A(x)\frac{\mathrm{d}f(z)}{\mathrm{d}z}, \quad \delta w = f(z)\frac{\mathrm{d}A(x)}{\mathrm{d}x}, \tag{A.8a}$$

where $f(z)$ and $A(x)$ are defined as

$$A(x) = \epsilon\,(C\cos(\alpha x) + S\sin(\alpha x)), \quad f(z) = \tfrac{1}{2} + \tfrac{1}{2}\sin\left(\frac{2\pi z}{L_z} - \frac{\pi}{2}\right). \tag{A.8b}$$

Here ϵ is a small value (taken to be 10^{-4} in the code) while C and S are constants between 0 and 1 (taken to be $\pm(2)^{-1/2}$, respectively). The chosen functional form (A.8) is justified as it is a simple model velocity field that is both incompressible and satisfies the flow boundary conditions on the channel walls.

The numerical simulations are seeded with the initial conditions (A.8). The resulting perturbation velocities are visualized and agree qualitatively with what is expected based on the no-slip boundary condition at the channel walls and the incompressibility condition—see figure A1. In particular, the perturbation velocity δu tends to zero at the channel walls across a narrow zone reminiscent of a boundary layer. A more precise test of the correctness of the numerical simulations can be obtained by considering the following quantities, which can be compared directly to the Orr–Sommerfeld theory:

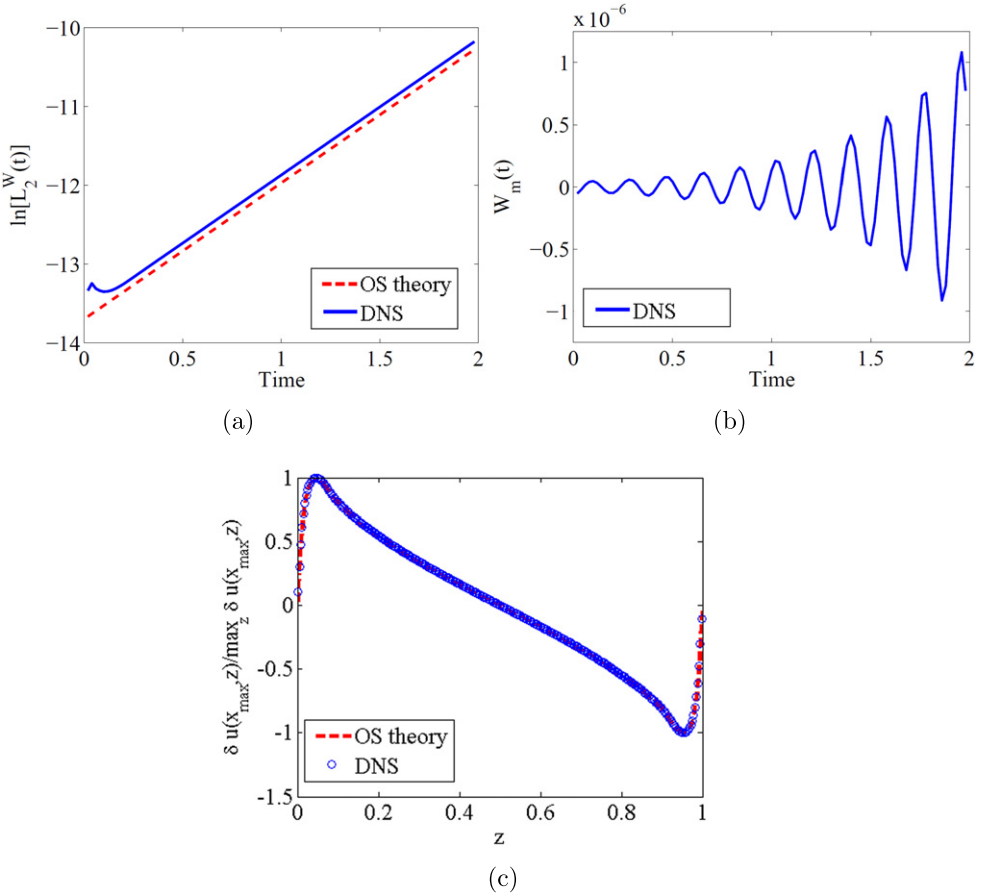

Figure A2. (a) Plot of $\ln[L_2^w(t)]$ with —— indicating the numerical result and - - - - the linear fit corresponding to $\lambda \approx 1.7293$; (b) plot of $W_m(t)$; (c) plot of $F(z)$ showing the agreement between the DNS and the OS analysis for the spatial structure of the perturbations. In (c) the circles are used to indicate the numerical values of $F(z)$ since otherwise the numerical and theoretical values would overlap completely.

$$L_2^w(t) = ||\delta w(x, z, t)||_2 = \left[\int_0^{L_x}\int_0^{L_z} (\delta w)^2 \mathrm{d}z\mathrm{d}x\right]^{1/2}, \tag{A.9a}$$

$$W_m(t) = \delta w\left(\frac{L_x}{2}, \frac{L_z}{2}, t\right), \tag{A.9b}$$

$$F(z) = \frac{\delta u(x_0, z, t)}{\max_{z\in[0,1]}\delta u(x_0, z, t)}, \tag{A.9c}$$

where x_0 corresponds to the location of the global maximum of $\delta u(x, z, t)$ in the whole flow domain. Theoretically, the L_2 norm of the δw perturbation ($L_2^w(t)$) should grow exponentially with time with a rate given by $\Re(\lambda)$, $W_m(t)$ should be an oscillating function with an exponentially growing envelope and period given by T and finally, $F(z)$ should be time-independent. The numerical results obtained are plotted in figure A2: very good agreement between the theory and the numerical simulation is obtained. We emphasize that such plots

are standard practice in the research-level literature whenever comparisons can be made between Orr–Sommerfeld theory and numerics [3, 38, 39].

In more detail, the predicted linear growth in $\ln(L_2^w(t))$ is evident from figure A2(a), where the growth rate is found to be $\lambda_S = 1.7293$. This agrees well with the predicted value (the two differing by an error of $\approx 4\%$) and decreased error is expected upon further grid refinement. Furthermore, the predicted behaviour of $W_m(t)$ is also observed in figure A2(b). The period of the oscillation is found to be $T_S \approx 0.186 \pm 0.004$, in excellent agreement with the predicted value of $T = 0.185$. The spatial structure of the flow is encoded in the function $F(z)$; excellent agreement in $F(z)$ between the numerical simulation and the theoretical calculation arising from the Orr–Sommerfeld eigenvalue analysis is evidenced in figure A2 (c).

References

[1] Scott D M, Bethune I, Ó Náraigh L and Valluri P 2013 Performance enhancement and optimization of the TPLS and dim two-phase flow solvers *HECToR dCSE Report* http://hector.ac.uk/cse/distributedcse/reports/tpls-dim/tpls-dim.pdf

[2] TPLS: High Resolution Direct Numerical Simulation of Two-Phase Flows. http://sourceforge.net/projects/tpls/

[3] Ó Náraigh L, Valluri P, Scott D M, Bethune I and Spelt P D M 2014 Linear instability, nonlinear instability and ligament dynamics in three-dimensional laminer two-layer liquid–liquid flows *J. Fluid Mech.* **750** 464–506

[4] Schneiderbauer S and Krieger M 2014 What do the Navier–Stokes equations mean? *Eur. J. Phys.* **35** 15020

[5] Kendl A 2008 Two-dimensional turbulence in magnetized plasmas *Eur. J. Phys.* **29** 911

[6] Fazarinc Z 1992 Discretization of partial differential equations for computer evaluation *Comput. Appl. Eng. Educ.* **1** 7385

[7] Heinricher A C and Rigden J S 1980 Numerical methods and physics instruction *Eur. J. Phys.* **1** 201

[8] Balay S *et al* 2015 PETSc Web page http://mcs.anl.gov/petsc

[9] NetCDF web page http://unidata.ucar.edu/software/netcdf

[10] Bethune I, Collis T, Ó Náraigh L, Scott D and Valluri P 2015 Developing a scalable and flexible high-resolution DNS code for two-phase flows *Proc. Int. Conf. on Parallel Computing (ParCo) 2015*in press

[11] Berselli L, Iliescu T and Layton W J 2006 *Mathematics of Large Eddy Simulation of Turbulent Flows* (Berlin: Springer)

[12] Abbott M B and Basco D R 1989 *Computational Fluid Dynamics—An Introduction for Engineers* (Harlow: Longman Scientific and Technical)

[13] Existence and smoothness of the Navier–Stokes equation http://claymath.org/sites/default/files/navierstokes.pdf

[14] Pope S B 2000 *Turbulent Flows* (Cambridge: Cambridge University Press)

[15] Lesieur M, Métais O and Comte P 2005 *Large-Eddy Simulations of Turbulence* (Cambridge: Cambridge University Press)

[16] Kim J, Moin P and Moser R 1987 Turbulence statistics in fully developed channel flow at low Reynolds number *J. Fluid Mech.* **177** 133–66

[17] Deardorff J W 1956 A numerical study of three-dimensional turbulent channel flow at large Reynolds numbers *J. Fluid Mech.* **41** 453–80

[18] Davidson P *et al* 2004 *Turbulence: An Introduction for Scientists and Engineers* (Oxford: Oxford University Press)

[19] Van Driest E R 2012 On turbulent flow near a wall *J. Aeronaut. Sci. (Institute of the Aeronautical Sciences)* **23** 1007–11

[20] Kang M, Fedkiw R P and Xu-Dong L 2000 A boundary condition capturing method for multiphase incompressible flow *J. Sci. Comput.* **15** 323–60

[21] Harlow F H *et al* 1965 Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface *Phys. Fluids* **8** 2182

[22] Chorin A J 1968 Numerical solution of the Navier–Stokes equations *Math. Comput.* **22** 745–62
[23] Boyd J P 2001 *Chebyshev and Fourier Spectral Methods* (New York: Dover)
[24] Garcia A L 2000 *Numerical Methods for Physics* 2nd revised edn (Englewood Cliffs, NJ: Prentice-Hall)
[25] Chi-Wang S and Stanley O 1988 Efficient implementation of essentially non-oscillatory shock-capturing schemes *J. Comput. Phys.* **77** 439–71
[26] Xu-Dong L, Stanley O and Tony C 1994 Weighted essentially non-oscillatory schemes *J. Comput. Phys.* **115** 200–12
[27] Durran D R 1991 The third-order Adams–Bashforth method: an attractive alternative to leapfrog time differencing *Mon. Weather Rev.* **119** 702–20
[28] Gropp W, Lusk E L and Skjellum A 1994 *Using Mpi: Portable Parallel Programming With the Message-Passing Interface* (*Scientific and Engineering Computation Series*) (Cambridge, MA: MIT Press)
[29] Yang X I A and Mittal R 2014 Acceleration of the Jacobi iterative method by factors exceeding 100 using scheduled relaxation *J. Comput. Phys.* **274** 695–708
[30] Saad Y 2003 *Iterative Methods for Sparse Linear Systems* (Philadelphia, PA: SIAM)
[31] Orr W M 1907 The stability or instability of the steady motions of a perfect liquid and of a viscous liquid: II. A viscous liquid *Proc. R. Irish Acad.* A **27** 69–138
[32] ARCHER—Technical assessment forms and notes http://archer.ac.uk/access/ta/
[33] Batten P, Goldberg U and Chakravarthy S 2004 Interfacing statistical turbulence closures with large-eddy simulation *AIAA J.* **42** 485–92
[34] Benocci C, van Beeck J P A J and Piomelli U 2006 *Large Eddy Simulation and Related Techniques: Theory and Applications (Von Karman Institute for Fluid Dynamics, 13–16 March 2006)*
[35] Dean R B 1978 Reynolds number dependence of skin friction and other bulk flow variables in two-dimensional rectangular duct flow *J. Fluids Eng.* **100** 215–23
[36] Schmid P J and Henningson D S 2012 *Stability and Transition in Shear Flows* vol 142 (New York: Springer)
[37] Orszag S A 1971 Accurate solution of the Orr–Sommerfeld stability equation *J. Fluid Mech.* **50** 689–703
[38] Valluri P, Náraigh L Ó, Ding H and Spelt P D M 2010 Linear and nonlinear spatio-temporal instability in laminar two-layer flows *J. Fluid Mech.* **656** 458–80
[39] Boeck T, Li J, López-Pagés E, Yecko P and Zaleski S 2007 Ligament formation in sheared liquid–gas layers *Theor. Comput. Fluid Dyn.* **21** 59–76